# Snowflake Naive Bayes and TPC-H

jarotek aff

December 2024

## 1 Introduction

Snowflake [1] is a cloud native data platform blabla. In the following we wish to investigate and performance test two implementations of Naive Bayes on Yelp Reviews, a pure SQL implementation and a UDTF. We will discuss performance and ease of implementation, as well as under-the-hood details.

Additionally we will conduct a standard TPC-H performance experiment with different configurations of Snowflake.

## 2 Methodology

### 2.1 Naive Bayes

The LATERAL FLATTEN function does a few key things:

Takes a semi-structured column (like an array or object) and creates a new row for each element Preserves the original row's context/data while expanding the nested elements Returns a virtual table with helpful metadata columns (value, seq, key, path, index)

### 2.2 TPC-H

For my TPC-H implementation I use data from the Snowflake-supplied database `SNOWFLAKE_SAMPLE_DATA`. I test TPC-H query 1, 5 and 18, which I copied from a website[1] and modified them to use Snowflake date functions.

Snowflake keeps a query history which, as everything else in Snowflake, can be queried on with SQL[2]. Every query gets hashed, and the query hash can subsequently be used to query the query history and analyze the performance of every execution of the query.

I have a Python script under my Github repository (`tpch/snowflake_tpch_generate.py`), which takes as program argument the number of iterations. It then generates a SQL script that can be copied into a SQL Worksheet in the snowflake client and executed.

The generation script puts a no-cache statement (`ALTER SESSION SET USE_CACHED_RESULT=FALSE;`) before each query. It then executes query 1, 5 and 18 on 4 sizes (extra small, small, medium, large) of warehouses, over scalefactor 1, 10, 100 and 1000. The warehouse is changed with the `USE WAREHOUSE` statement, and the scalefactors are changed with the `use schema` statement, (e.g. `use schema snowflake_sample_data.tpch_sf1000`).

To distinguish between the 3 queries on the different scale factor and warehouse settings, I augment the queries to select a string literal to force a unique query hash, e.g.

```
SELECT 'BISON_WH_XS_Q1_SF1000', (...)
```

[1] https://docs.deistercloud.com/content/Databases.30/TPCH\%20Benchmark.90/Sample\%20querys.20.xml?embedded=true\#7b15827ccb57cbcec68be7a26953590c

[2] https://docs.snowflake.com/en/user-guide/performance-query-exploring#query-track-the-average-performance-of-a-query-over-ti

I have 48 distinct queries (3 queries * 4 warehouses * 4 scalefactors). I collect the query hashes manually from the query history, and I can then analyse the execution time with:

```
    SELECT
    query_parameterized_hash,
    COUNT(*) as query_count,
    AVG(total_elapsed_time) as mean_elapsed_time,
    STDDEV(total_elapsed_time) as std_elapsed_time,
    MAX(total_elapsed_time) as max_elapsed_time,
    MIN(total_elapsed_time) as min_elapsed_time,
    AVG(bytes_scanned)
FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY
WHERE query_parameterized_hash IN ( ... )
AND DATE_TRUNC('day', start_time) >= CURRENT_DATE() - 30
GROUP BY query_parameterized_hash
ORDER BY mean_elapsed_time DESC;
```

I execute the queries 20 times across a whole day.

# 3 Results and Discussion

## 3.1 TPC-H

I start my analysis by investigating correlation between megabytes scanned and execution time.

| Warehouse | Query | Correlation |
|---|---|---|
| BISON_WH_L | Q1 | 0.999 |
| BISON_WH_L | Q18 | 1.000 |
| BISON_WH_L | Q5 | 1.000 |
| BISON_WH_M | Q1 | 1.000 |
| BISON_WH_M | Q18 | 1.000 |
| BISON_WH_M | Q5 | 1.000 |
| BISON_WH_S | Q1 | 1.000 |
| BISON_WH_S | Q18 | 1.000 |
| BISON_WH_S | Q5 | 1.000 |
| BISON_WH_XS | Q1 | 1.000 |
| BISON_WH_XS | Q18 | 0.999 |
| BISON_WH_XS | Q5 | 1.000 |

Table 1: Pearson correlation between MB scanned and execution time by warehouse and query type (rounded to three decimals)

In table 1 we see that there is a definitive positive correlation between MB scanned and execution time. Given that we group by warehouse and query type, we end up checking correlation between the different scale factors and with a factor of 10 between each scale factor it is perhaps not so strange that the MB scanned end up having a strong correlation.

In figure 1 we see that query 18 scales the most linearly, whereas the other are penalized less by scale factor.

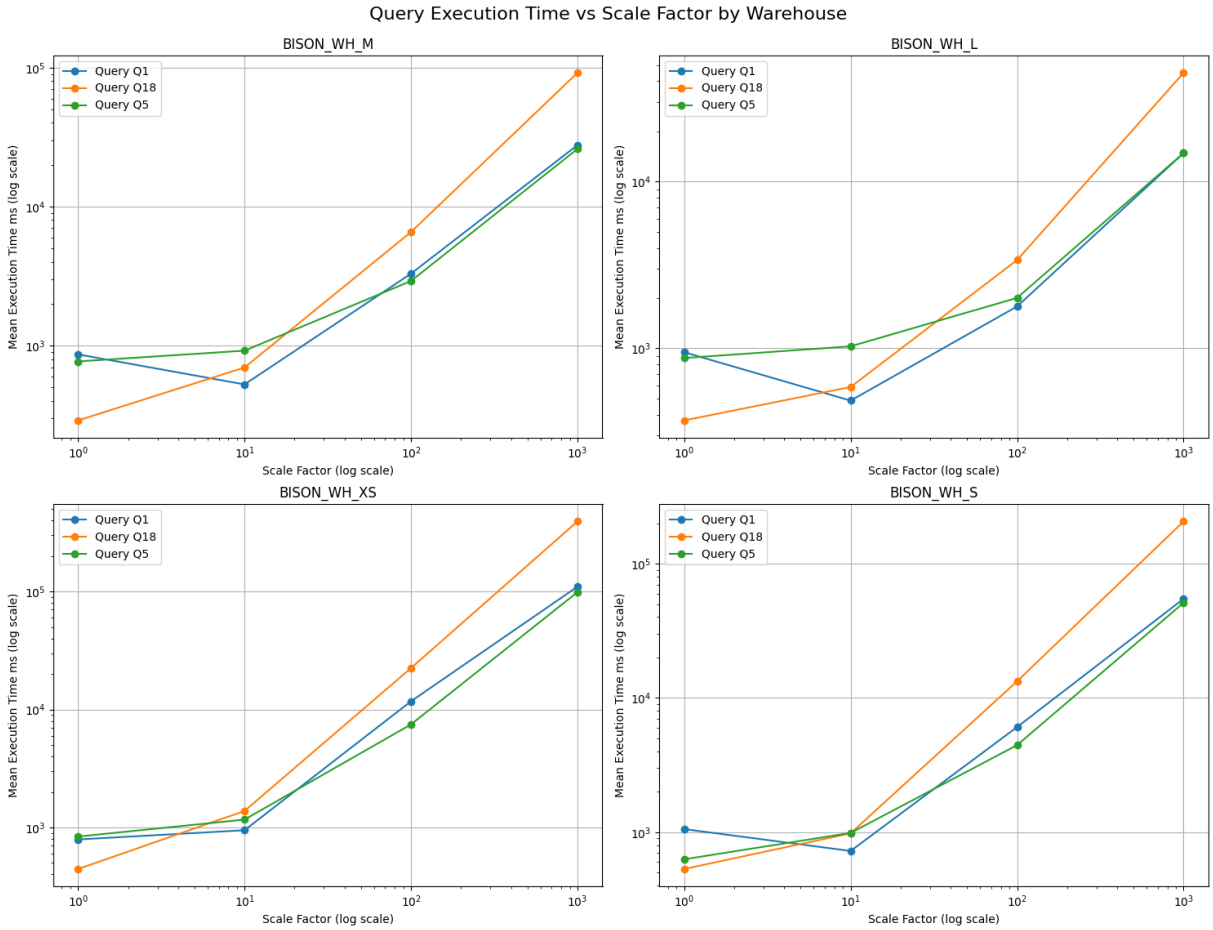


Figure 1: Log log plots of the queries under different scale factors for each warehouse size.

### 3.2 Naive Bayes

#### 3.2.1 SQL

Overall Accuracy: 38.54%

## 4 Conclusion

# References

[1] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. The snowflake elastic data warehouse. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 215–226, New York, NY, USA, 2016. Association for Computing Machinery.